MACAON :
visite guidée

Alexis Nasr

23 novembre 2017

Table of contents

Multi Column File (`mcf`)

Format de fichier qui permet de représenter les données textuelles et les annotations qui y sont attachées

- format “en colonne” :
  - chaque ligne correspond à une unité textuelle minimale (token)
  - chaque colonne correspond à un attribut du token
- les colonnes sont séparées les unes des autres par une tabulation
- le nombre de colonnes est illimité
- l'interprétation de chaque colonne est décrite dans un fichier `mcd` (Multi Column Description)
- les lignes commençant par ## sont ignorées

mcf : exemple

```
La          det     le          1    det         0
diane       nc      diane       1    suj         0
chantait    v       chanter     0    root        0
dans        prep    dans       -1    mod         0
la          det     le          1    det         0
cour        nc      cour       -2    obj         0
des         prep    des        -1    dep         0
casernes    nc      caserne    -1    obj         0
.           poncts  .          -6    eos         1
Et          coo     et          0    root        0
le          det     le          1    det         0
vent        nc      vent        3    suj         0
du          prep    du         -1    dep         0
matin       nc      matin      -1    obj         0
soufflait   v       souffler   -5    dep_coord   0
sur         prep    sur        -1    mod         0
les         det     le          1    det         0
lanternes   nc      lanterne   -2    obj         0
.           poncts  .          -9    eos         1
```

Multi Column Description (`mcd`)

Un fichier `mcd` associe une étiquette à chaque colonne d'un fichier `mcf`

- Chaque ligne du fichier `mcd` décrit une colonne du fichier `mcf`.
- chaque ligne du fichier `mcd` est formée de quatre colonnes
  1. un entier indiquant la colonne décrite
  2. l'étiquette correspondant à la colonne
  3. le type de la valeur s'y trouvant
     trois types sont définis :
     - `VOCAB` indique que les valeurs sont des symboles
     - `INT` indique que les valeurs sont des entiers
     - `EMB` indique que les valeurs sont des vecteurs de réels
  4. le fichier où sont stockés les vecteurs dans le cas où le type est `EMB`

Etiquettes

- Une étiquette permet de donner un nom explicite à une colonne
- Elles permettent d'accéder au contenu de la colonne à l'aide de *Word Features*
- Liste des étiquettes définies :
  - `FORM` la forme du token
  - `CPOS` sa partie de discours grossière
  - `POS` sa partie de discours
  - `LEMMA` son lemme
  - `FEATS` d'autres traits (en général morphologiques)
  - `GOV` la position relative de son gouverneur (-n indique que le gouverneur se trouve n tokens à gauche, n indique qu'il se trouve n tokens à droite)
  - `LABEL` sa fonction syntaxique
  - `SENT_SEG` sa position dans la phrase (1 dernier mot de la phrase, 0 autre)
  - `A ...Z` autres étiquettes permettant de représenter des informations utiles (durée d'un token, locuteur, ...)

mcd exemple

- un extrait du fichier mcf

```
La          det     le          1     det          0
diane       nc      diane       1     suj          0
chantait    v       chanter     0     root         0
```

- le fichier mcd correspondant

```
1    FORM      VOCAB _
2    POS       VOCAB _
3    LEMMA     VOCAB _
4    GOV       INT   _
5    LABEL     VOCAB _
6    SENT_SEG  INT   _
```

- C'est le mcd par défaut (appelé wplgfs)

Features simples

- Fonctions permettant d'accéder au contenu d'une configuration
- Quatre types de Features simples :
  1. Word Features, attributs des tokens
  2. Syntactic Features, structure syntaxique déjà construite
  3. Distance Features, distance entre certains mots
  4. Configurational Features, autres informations (non linguistiques) sur la configuration
- Toutes les Features simples sont codées en dur dans le fichier `feat_fct.c`

Word features

Elles sont composées :

- d'une fonction d'adresse, qui indique à quel token on fait référence dans une configuration :

b0	token courant dans le buffer	
b1	token directement à droite de	b0
b2	token directement à droite de	b1
b3	token directement à droite de	b2
bm1	token directement à gauche de	b0
bm2	token directement à gauche de	bm1
bm3	token directement à gauche de	bm2
s0	token au sommet de la pile	
s1	token directement sous	s0
s2	token directement sous	s1
s3	token directement sous	s2

Word features

- d'une fonction d'attribut qui accède à un attribut du token :

`f`	forme	`FORM`
`l`	lemme	`LEMMA`
`c`	partie de discours grossière	`CPOS`
`p`	partie de discours	`POS`
`m`	traits morphologiques	`FEATS`
`s`	fonction syntaxique	`LABEL`
`A ...Z`	autres attributs	`A ...Z`

- Une word Feature est la concaténation d'une fonction d'adresse et d'une fonction d'attribut, par exemple : `b0f`

Syntactic Features

ldep_s0r	fonction du dépendant le plus à gauche de s0
rdep_s0r	fonction du dépendant le plus à droite de s0
ldep_s1r	fonction du dépendant le plus à gauche de s1
rdep_s1r	fonction du dépendant le plus à droite de s1
ndep_s0	nombre de dépendants de s0

Distance Features

- `dist_s0_b0` linear distance between s0 and b0

Configurational Features

sh	number of elements in the stack
t1	code de la transition menant à la configuration courante
t2	code de la transition précédant t1
t3	code de la transition précédant t2
t4	code de la transition précédant t3
mvt0	code de la première transition de meilleur poids
mvt1	code de la deuxième transition de meilleur poids
mvt2	code de la troisième transition de meilleur poids
mvt3	code de la quatrième transition de meilleur poids
delta1	score(mvt0) - score(mvt1)
delta2	score(mvt0) - score(mvt2)
delta3	score(mvt0) - score(mvt3)

Feature Model (`fm`)

- Un fichier `fm` définit une collection de Features qui vont permettre de décrire une configuration
- Ces Features sont utilisées par le classifieur pour prédire le prochain mouvement
- Chaque ligne d'un fichier `fm` définit une Feature
- Deux types de Features sont distingués :
  - Les Features simples, choisies dans la liste des Features simples
  - Les Features complexes, composées de plusieurs Features simples disposées sur une même ligne, séparées par un espace

fm : Exemple

```
b0f
s0l
s0p b0p
```

1. Feature simple `b0f` (forme du mot courant dans le buffer)
2. Feature simple `s0l` (lemme du mot se trouvant au sommet de la pile)
3. Feature complexe `s0p b0p`, sa valeur est composée de la partie de discours du mot situé au sommet de la pile et celle du mot courant dans le buffer (par exemple `n_v`)

- Ambiguïté classique, si on a `b0f = manger`
  - `b0f` est un identifiant de features, noté Feature
  - `manger` est une valeur de feature, notée feature

`dico`

Ensemble de couples :

(symbole (chaîne de caractère), code (entier))

- La première ligne indique le nom du `dico`
- La seconde le nombre d'entrées qui le composent
- Les autres, les chaînes de caractères qui composent le `dico`
- La première chaîne a pour code `0`, la seconde `1` ...
- On peut représenter plusieurs `dico` dans un seul fichier en utilisant le séparateur `##_DICO_END_##`

dico

```
FORM
4
Certes
,
rien
ne
##_DICO_END_##
POS
3
adv
ponctw
pro
```

- Deux dico dans un même fichier
- FORM composé de 4 entrées associées respectivement aux entiers 0, 1, 2 et 3
- POS composé de 3 entrées associées respectivement aux entiers 0, 1 et 2

Class Features File (`cff`)

- Format utilisé pour entraîner les classifieurs
- Chaque ligne est constituée d'une liste d'entiers
- La première colonne correspond à un code de classe (des entiers consécutifs commençant à 0)
- Les autres colonnes correspondent à des codes de features, décrits, par exemple, dans un fichier `dico`.

Class Feature Weights (`cfw`)

- Format binaire contenant les poids des couples

  (classe, feature)

- Structure du fichier :
- `F` nombre de features (`int`)
- `C` nombre de classes (`int`)
- `F` tableaux de `C` `float` (les poids associés aux features)
- Exemple :

```
4
3
1.23 4.61 -2.45 5.28
2.41 -4.55 7.29 3.33
-5.34 6.33 6.40 2.34
```

maca_trans_parser_arc_eager_mcf2cff

entrées :

- un fichier `mcf` annoté en syntaxe (colonnes `GOV` et `LABEL`)
- un fichier `fm` décrivant les features utilisées pour la prédiction
- un fichier `mcd` décrivant la structure du fichier `mcf` (optionnel)

sorties :

- un fichier `cff` contenant les données pour l'apprentissage du classifieur
- un fichier `dico` contenant les vocabulaires correspondant aux features utilisées et le vocabulaire des features du classifieur

perceptron_train

entrées :

- un fichier `cff`

sorties :

- un fichier `cfw`

description :

- Calcule un poids pour tout couple (couple, feature).
- Les poids sont stockés dans le fichier de sortie.

maca_trans_parser_arc_eager

entrées :

- un fichier `mcf` contenant le texte à analyser
- un fichier `fm` contenant les modèles features
- un fichier `cfw` contenant les poids

sorties :

- un fichier `mcf` correspondant au fichier `mcf` d'entrée plus trois colonnes : `GOV`, `LABEL`, `SENT_SEG`

Dépots GIT

Le projet macaon est réparti sur deux dépôts GIT :

- `macaon2` contient le code des différents modules
  `https://gitlab.lif.univ-mrs.fr/alexis.nasr/macaon2`
- `maca_data2` contient les données primaires ainsi que les procédures de compilation des modèles à partir des données primaires
  `https://gitlab.lif.univ-mrs.fr/alexis.nasr/maca_data2`

macaon2

Installation :

```
git clone https://gitlab.lif.univ-mrs.fr/alexis.nasr/macaon2.git
cd macaon2
mkdir build
cd build
cmake ..
make
sudo make install
```

macaon2 structure

- maca_common
  word.c word_buffer.c mcd.c
- perceptron
  cf_file.c perceptron.c feature_table.c (cfw)
- maca_tokenizer
- maca_lexer
- maca_lemmatizer
- maca_trans_parser
  config.c movements.c feat_fct.c
- maca_tools

maca_data2 installation

```
git clone https://gitlab.lif.univ-mrs.fr/alexis.nasr/maca_data2.git
cd maca_data2
cd tools
make
cd ../fr
make
```

dans .bashrc, ajouter :

```
export MACAON_DIR=/home/xxx/maca_data2
```

maca_data2 structure

- $n + 3$ répertoires :
  - tools
  - data données primaires (ftb, ptb ...)
  - makefiles nécessaires pour compiler les modèles
  - un répertoire par *langue* (fr, en ...)
- les répertoires des langues ont tous la même structure :
  - data données pour l'apprentissage, construites à partir des données primaires
    - treebank (train.mcf, dev.mcf, test.mcf)
    - morpho-lexicon (fplm, fP)
  - maca_tokenizer
  - maca_lexer
  - maca_trans_tagger
  - maca_trans_parser
  - bin
  - eval (results)

Utilisation

```
echo "Jean aime Marie." | maca_tokenizer | maca_lexer
| maca_trans_tagger | maca_lemmatizer | maca_trans_parser
Jean  np     Jean   1 suj 0
aime  v      aimer  0 root 0
Marie np     Marie -1 obj 0
.     poncts .     -2 ponct 1
```

- langue choisie par défaut : `fr`
  (pour changer : `-L en`)
- `mcd` choisi par défaut : `wplgfs`
  (pour changer : `-C fichier.mcd`)

Le mode debug (option -d)

```
**************************   **************************   **************************
[ ]                            [ 1 2]                         [ 1]
[0:Jean] 1:aime 2:Marie 3:.    0:Jean 1:aime 2:Marie [3:.]    1:aime 2:Marie 3:.
0 SHIFT        171.2102        0 REDUCE       203.7658        0 ROOT      151.8154
1 RIGHT suj     45.9099        1 RIGHT abbrev  62.4709        1 RIGHT suj  31.7654
2 RIGHT mod     35.5334        2 RIGHT ponct   35.3505        2 REDUCE     25.0480
**************************   **************************
[ 0]                           [ 1]
0:Jean [1:aime] 2:Marie 3:.    0:Jean 1:aime 2:Marie [3:.]
0 LEFT suj      88.2239        0 RIGHT ponct  151.4706
1 LEFT mod      31.1564        1 RIGHT abbrev  26.5903
2 LEFT ato      23.3244        2 RIGHT aff     25.8576
**************************   **************************
[ ]                            [ 1 3]
0:Jean [1:aime] 2:Marie 3:.    1:aime 2:Marie 3:.
0 SHIFT        172.5921        0 EOS          218.4483
1 RIGHT abbrev  26.6208        1 REDUCE       112.4713
2 RIGHT aff     25.3750        2 ROOT          32.4288
**************************   **************************
[ 1]                           [ 1 3]
0:Jean 1:aime [2:Marie] 3:.    1:aime 2:Marie 3:.
0 RIGHT obj     60.3574        0 REDUCE       151.9829
1 RIGHT suj     46.2737        1 EOS           72.6095
2 RIGHT ats     33.3195        2 ROOT          27.9275
```

Le décodeur

```
c = config_new(f, mcd, lookahead);
while(!config_is_terminal(c)){
    config2feat_vec_cff(fm, c, dico_features, fv, LOOKUP_MODE);
    mvt_code = feature_table_argmax(fv, ft, &max);
    mvt_type = movement_parser_type(mvt_code);
    mvt_label = movement_parser_label(mvt_code);
    result = 0;
    switch(mvt_type){
    case MVT_PARSER_LEFT :
      result = movement_parser_left_arc(c, mvt_label); break;
    case MVT_PARSER_RIGHT:
      result = movement_parser_right_arc(c, mvt_label); break;
    case MVT_PARSER_REDUCE:
      result = movement_parser_reduce(c); break;
    case MVT_PARSER_ROOT:
      result = movement_parser_root(c, root_label); break;
    case MVT_PARSER_EOS:
      result = movement_parser_eos(c); break;
    case MVT_PARSER_SHIFT:
      result = movement_parser_shift(c);
    }
    if(result == 0)
      result = movement_parser_shift(c);
}
```

word

```
typedef struct _word {
  /* array containing the codes corresponding to the different word features */
  int wf_array[MCD_WF_NB];
  /* the string corresponding to the actual line in the corpus file */
  char *input;
  /* does the form begin with an uppercase character */
  int U1;
  /* pos tags that this form can have (represented as a boolean string) */
  int signature;
  int label;
  char *form;
  int index;
  int is_root;
} word;
```

mcd

```
typedef struct {
  /* number of columns in the mcd file */
  int nb_col;
  /* in which column are the word features
  (MCD_WF_FORM, MCD_WF_LEMMA ...) represented */
  int wf2col[MCD_WF_NB];
  /* array containing the word feature that correspond to each column */
  int *wf;
  /* a string version of array word feature */
  char **wf_str;
  /* array containing the representation mode of every column
  (integer, vocabulary, embedding, NULL) */
  int *representation;
  /* array containing the file in which the different values
  for a columnn is represented */
  char **filename;
  /* array containing the dico corresponding to each column (NULL if no file) */
  dico **dico_array;
  /* array containing the word embedding structure corresponding
  to each column (NULL if no file) */
  word_emb **word_emb_array;
} mcd;
```

word_buffer

```
typedef struct {
  /* size of the array used to store words */
  int size;
  /* number of words in the buffer */
  int nbelem;
  /* number of words between the current word and the last word of the buffer */
  int lookahead;
  /* position of the current word */
  int current_index;
  /* array to store words */
  word **array;
  /* file to read the words from */
  FILE *input_file;
  /* mcd describing the format of input_file */
  mcd *mcd_struct;
} word_buffer;
```

config

```
typedef struct {
  /* the stack */
  stack *st;
  /* the buffer */
  word_buffer *bf;
  /* movement sequence that led to this configuration */
  mvt_stack *history;
  int mvt_chosen;
  vcode *vcode_array;
} config;
```